SYSTEM AND METHOD FOR SOURCE-DRIVEN FORM-INDEPENDENT DYNAMIC CONTENT RESOLUTION

FIELD OF THE INVENTION

This invention relates to the field of source-driven form-independent dynamic content resolution over a network for the use of practicing a business method over a network.

BACKGROUND OF THE INVENTION

The explosion of the Internet and the World Wide Web has ushered in an era of unprecedented integration of technology and people. A resounding "yes" to the question of "are you connected" reflects the ubiquitous permeation of the online experience round the globe. Grandparents can now monitor their grandchildren continents apart over the Internet. Individuals to governments to industrial conglomerates all tap into a supply chain infrastructure wrapped around the world many times over, all glued together through Internet technology.

A chain reaction from this explosion is a parallel boom in the head-numbing choices in Internet-related devices and applications people have access to, everything from digital computers to hand-held devices to wearable computing to things that think (Things That Think

Express Mail: EV001687886US
Date of Deposit:November 15, 2001

Consortium headed by Massachusetts Institute of Technology Media Lab) and their related applications.

The concept of separation of content from form has become the defacto standard solution in Internet technology to the challenge of enabling the exchange of information over the Internet through this multitude of devices and applications. Related to interfaces, form means the "look and feel" of an interface while content means the data delivered through the "look and feel". The online dictionary at www.m-w.com defines form as, "the shape and structure of something as distinguished from its material", or "the essential nature of a thing as distinguished from its matter." The "material" and "matter" refer to the content. Not long ago Mosaic was the only graphic user interface to the Internet, and static HTML pages represented the sole source which delivers both the form and the content to users through a graphic user interface over the Internet.

No longer do graphic designers need to understand database API and vice versa, at least in principle. The separation and componentization of, and therefore the efficient utilization of non-overlapping expertise are what separating content and form intends to bring.

STATEMENT OF PROBLEMS WITH THE PRIOR ART

While the concept of separation of content and form is a valid one, in reality, content and form cannot stay separate forever; they have to be integrated eventually in a unified presentation. Here lies the new challenge: on one hand, separation of content and form has led to an exponential growth in the fields of both content and form generation now that the content experts such as

database programmers and the form experts such as the graphic designers can work separately of each other without learning each other's skills; on the other, it has given birth to a daunting challenge of putting content and form back together, a vacuum over which neither the content nor the form experts claim responsibility. This vacuum has to be filled because at the end you can separate content and form as much as you want, but neither can stand on its own.

Many approaches to this new paradox include standardization which forces both the content and form created to conform to an agreed-upon set of rules and specification for integration. Recent flourishing of markup languages such as XML and the rest of the alphabet soup is an example of this trend. In this approach, both the content and form experts need to understand the standard and conform to it, added labor which did not exist before and which restricts freedom and flexibility of their respective work. Often in these standards, form dominates content in reality, while in concept content has a greater degree of flexibility than form. After all, a name, the content, can be delivered virtually anywhere, but the form, whether it be visual, audio, digital, vocal, etc., through which it is delivered is often severely limited by circumstances.

Another approach is to surrender control to a third-party application such as plug-ins which handles its own content and form processing. Netscape plug-ins are a well-known example. Recently, Microsoft has announced that it will pull the plug on its support of plug-ins, leaving ActiveX controls, the Microsoft-equivalent of plug-ins the only choice for its Internet Explorer shipped with Windows XP. In essence, plug-ins, ActiveX controls and the like offer a bypass of the separation of content and form altogether, not a solution.

Further efforts now go into dynamic content generation. Examples include US patents US 5,835,712, Client-server system using embedded hypertext tags for application and database development, US 6,256,032, Method and apparatus for organizing and processing information using a digital computer, US patent 5,940,834, Automatic web page generator, and US patent 5,983.227, Dynamic page generator.

These efforts are rooted behind the thinking that the generation of content, new content which did not exist prior to these efforts is necessary, therefore they focus on coming up with novel approaches to generate the new content. In the process, these efforts blur the line differentiating content from form, force content and form to conform specifications of a particular approach, and/or generate new content specifically for a particular approach, thus defeating the key purposes behind the separation of content and form: 1. Form experts and content experts could work separately without learning each other's skills. 2. Form and content thus created are reusable for different presentations based on different approaches.

On the other hand, this patent is based on a simple but different philosophy, that of dynamic integration of content and form, which at the end aims for a unified presentation of both, but requires nothing more than the placements of previously created content and form with no new content being generated specifically for this approach. This approach emphasizes the separation of content and form by maintaining control over the placement decision of form and content in a unified presentation and yet allowing separate requirements on content and form to drive that decision, for example on how they are to be accessed, retrieved and inserted separately based on their own requirements.

OBJECTS OF THE INVENTION

An object of this invention is an improved business method for source-driven form-independent dynamic content resolution over a network.

An object of this invention is an improved business method for insert statement parsing for use in source-driven form-independent dynamic content resolution over a network.

An object of this invention is an improved business method for content source identification for use in source-driven form-independent dynamic content resolution over a network.

An object of this invention is an improved business method for content source access for use in source-driven form-independent dynamic content resolution over a network.

An object of this invention is an improved business method for content insertion for use in source-driven form-independent dynamic content resolution over a network.

SUMMARY OF THE INVENTION

The present invention is a computer system and method for dynamic content resolution. The invention comprises one or more source program files and one or more insert statements inserted into the source program files. The insert statements having one or more logical condition

statements with one or more logical parameters and one or more state statements. An insert statement parser determines a state of the condition statement, selects one of the state statements associated with the state, and parses the selected state statement into one or more content source indicators. A content source identification process uses one or more of the content source indicators to determine a content source type and one or more access instructions. A content source access process uses the access instructions to access a source object. A content insertion process replaces the insert statement with the source object in the source program file.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing and other objects, aspects and advantages will be better understood from the following detailed description of preferred embodiments of the invention with reference to the drawings that include the following:

Figure 1 is a block diagram of a source driven form-independent dynamic content resolution system also known as a content composor.

Figure 2 is a block diagram of an example insert statement.

Figure 2A is block diagram of a nested content source indicator.

Figure 3 is a block diagram of a content source access map.

Figure 4 is a block diagram of the insert statement parser rules.

Figure 5 is a flow chart of the insert statement parser process.

Figure 6 is a flow chart of the content source identification process.

Figure 7 is a flow chart of the content source access process.

Figure 8 is a flow chart of the content insertion process.

Figure 9 is an example HTML program source file.

Figure 10 is the resolved source of the example HTML program source file.

Figure 11 is a rendering of the resolved source of the example HTML program source file.

DETAILED DESCRIPTION OF THE INVENTION

Figure 1 is a block diagram of a source driven form-independent dynamic content resolution system, also known as a content composer 100. Figure 1 contains a computer system 130, which is connected to a network 140 through a network connection 135, Direct Access Storage Devices (DASD) 170, and Live Content Sources 140. Live Content Sources 140 could include

temperature or weight sensors, electronic instruments, microphones, video cameras, still digital cameras, etc.

The computer system 130 contains one or more Central Processing Units (CPUs) 165 and one or more Memories 145. The computer system 130 is running an application program 125, which contains the content composor 100. The content composor 100 accesses one or more memories 145.

The content composor 100 contains four processes: an insert statement parser process 500 further described in Figure 5, a content source identification process 600 further described in Figure 6, a content source access process 700 further described in Figure 7, and a content insertion process 800 further described in Figure 8.

A source program file 900 is an input to the insert statement parser process 500. A source program file contains form information, which is defined in the background of this invention, and novel insert statements 205 further described in Figure 2. Figure 9 further describes an example source program file 900 referred to as HTML source program file, which contains Hypertext Markup Language (HTML) scripts and optionally insert statements 205. In Figure 9, form information refers to instructions conforming to HTML syntax on how to render specific forms such as table, column, font, color, size, etc. Figure 10 further describes the resolved HTML source, i.e. with the insert statement 205 being replaced by content, of the example HTML source

program file in Figure 9. Figure 11 further describes the graphic rendering of that resolved HTML source in Figure 10.

The insert statement parser process 500 starts by locating insert statements 205 within a source program file 900. The structure of an insert statement 205 is further described in Figure 2, an insert statements tructure block diagram 200. The insert statement parser process 500 then parses the insert statement 205 into its components, namely one or more content source indicators 210. The insert statement parser rules 400 are further described in Figure 4. The one or more content source indicators 210, which are further described in Figure 2, are arranged hierarchically into one or more state statements 225 and one or more condition statements 245. As an alternative, one or more additional insert statement 205 could also be nested within a content source indicator 210 and is further described in Figure 2A.

Once a content source indicator 210 is parsed, the content source identification process 600 then takes the content source indicator 210 as input and identifies the content source object 305 it represents through the use of a content source access map 300, which is further described in Figure 3. A content source object 305 is one or more of the following: a mass data storage application, an HTML program source file, a static file, a memory access, a multimedia data file, text file, XML file, binary data file, remote file, and live content. Examples of live content include data from live content source objects 305 such as temperature or weight sensors, electronic instruments, microphones, video cameras, still digital cameras, etc.

Once a content source object 305 is identified, the content source access process 700 locates its associated access instructions and then accesses it. Finally if access is successful, the content insertion process 800 retrieves the content from the content source object 305 and inserts it in place of the insert statement 205, either by reference or value.

Figure 2 is a block diagram of an example insert statement 205. The following shall describe the example insert statement 205 from left to right starting with the keyword, INSERT. INSERT is the insert statement indicator 250, which indicates the start of an insert statement 205 to the insert statement parser process 500. The open and close brackets before INSERT and at the end of the insert statement, along with the hyphens, periods and spaces are part of the preferred rules 400 with which the insert statement parser process 500 parses the insert statement 205. Figure 4 further describes these preferred rules 400.

Following INSERT are two state statements 225, state statement 1 (225) and state statement 2 (225). Each state statement 225 has one or more content source indicators 210 and the content source indicators 210 are in a hierarchy of indicators. The state statements 225 is this example contain two or three content source indicators 210. One of the content source indicators 210 of each state statement 225 is a content source type indicator 230. A content source type indicator 230 is a content source indicator 210 which further indicates the type of a source object 305.

In this example, state statement 1 (225) contains the keywords, DB , PERSON and FIRST_NAME respectively from left to right. Each keyword is a content source indicator 210. In this example, DB, the first content source indicator 210 from left in state statement 1 (225), is a

content source type indicator 230, associated with a content source object 305, mass data storage application or a database application more specifically. The hierarchy of content source indicators 210 of, from left to right in state statement 1 (225), DB, PERSON and FIRST_NAME is associated with a content source object 305, its content source type, its location as well as location of its content access and retrieval instructions.

Similarly, state statement 2 (225) contains the keywords, HTML, DEFAULT_PERSON, and DEFAULT_NAME respectively from left to right, each a content source indicator 210. In this example, HTML, the first content source indicator 210 from left in state statement 1 (225) is a content source type indicator 230, associated with a content source object 305, an HTML source program file in this case. Figure 3 further describes the hierarchy of content source indicators 210 in each of the state 225 and condition statements 245 and the association to its content source object 305.

Following the two state statements 225 in this example is one logical condition statement 245. This logical condition statement 245 contains the keywords, TRANS and PERSON respectively from left to right, each content source indicator 210. In this example, TRANS, the first content source indicator 210 from left in this logical condition statement 245, is a content source type indicator 230. In the preferred embodiment, TRANS refers to memory 145 which contains context 515 information for the content composor 100. Figure 3 further describes the hierarchy of content source indicators 210 in each of the state 225 and condition statements 245 and the association to its content source object 305.

Notice that none of the content source indicators 210 in Figure 2 described above is nested. The following shall describe a nested content source indicator 210.

Figure 2A is block diagram of a nested content source indicator 210. In the preferred embodiment, the nested content source indicator 210 could be from a hierarchy of content source indicators 210, which could be from a state 225 or a logical condition statement 245 within an insert statement 205.

Starting from left to right, following the keywords DB and COMPANY and before OWNER is an insert statement 205 contained between a pair of parantheses. The pair of parentheses is the preferred nested content source indicator identifier 425 which indicates the existence of a nested content source indicator 210. In this example, the pair of parentheses is part of the preferred rules followed the insert statement parser process 500 further described in Figure 5.

The nested insert statement 205 in this example, starting with the keyword INSERT, contains two state statements 225, and one logical condition statement 245. State statement 1 (225) contains keywords HTML and CURRENT_ACTNO from left to right, each a content source indicator 210 with HTML being a content source type indicator 230. Similarly, state statement 2 (225) contains keywords TRANS and ACTNO from left to right, each a content source indicator 210 with TRANS being a content source type indicator 230. The logical condition statement 245 contains keywords TRANS and ACTIVE from left to right, each a content source indicator 210 with TRANS being a content source type indicator 230. The description above on the structure of an insert statement in Figure 2 applies here.

The insert statement parser process 500, the content source identification process 600, the content source access process 700 and the content insertion process 800 then process and resolve this nested insert statement 205, the structure of which is described in Figure 2. As an example, resolving the nested content source indicator could yield

-DB.COMPANY.A454.OWNER,

with A454 replacing the nested insert statement,

INSERT -HTML.CURRENT_ACTNO -TRANS.ACTNO -TRANS.ACTIVE

Notice that none of the content source indicators 210 of the nested insert statement 205 in this example is nested. Otherwise, the description above on the nested content source indicator 210 applies recursively until none of the content source indicators 210 of an insert statement 205 is nested. Each nested insert statement 205 would then be resolved and replaced by content from a content source object 305. A path refers to the recursive calls processing the nested insert statements.

Figure 3 is a block diagram of an example content source access map 300. A content source access map 300 is a table which associates hierarchies of content source indicators 210 with content source objects 305 and content source object information 310. The content source object

information 310 includes content source type information, optional location information and optional content access and retrieval instruction location information.

The content source indicators 210 column of the content source access map 300 contains a hierarchy of content source indicator 210 keywords, such as DB, PERSON and FIRST_NAME in the first cell from the top of the middle column, each keyword a content source indicator 210. The first row of the example content source access map 300 contains the keywords DB, PERSON and FIRST_NAME. These keywords correspond to the keywords of state statement 1 (225) in Figure 2.

The content source objects 305 column contains cells grouped from top to bottom, corresponding to cells in the middle column. Each cell contains a content source object 305 associated with the hierarchy of content source indicators in the corresponding cell of the middle column. From top to bottom, these content source objects 305 include mass data storage application, HTML program source file, static file, memory access and multimedia data file.

The content source object information 310 column contains cells grouped from top to bottom, corresponding to cells in the middle column. Each cell contains information on a content source object 305 associated with the hierarchy of content source indicators 210 in its corresponding cell in the middle column. The information on a content source object 305 includes its content source type, its location as well as location of its content access and retrieval instructions 310. Notice that the information 310 is grouped hierarchically in parallel to the hierarchy of content source indicators 210.

Take the first cell for example, DLL SXDB OPENFN contains the access instruction to the mass data storage application referred to by the content source indicator 210, DB. DLL indicates the type of the access instruction which is Dynamically Linked Library (DLL) in this case. SXDB indicates the location where the access instruction is at. SXDB is a library file which is compiled and linked from source codes written in the computer language C. OPENFN indicates the function within SXDB to be called in order to gain access to the mass data storage application. The execution details such as compiling and linking of the source codes written in C are well understood as prior arts.

The following line, DLL SXDB_PERSON SXDBFINDPERSON contains the access instruction to the person table within the mass data storage application. PERSON is the content source indicator referring to the person table. Finally, DLL SXDB_PERSON SXDBQUERYFNFORPERSON contains the access instruction to the first name entry within the person table within the mass data storage application. FIRST_NAME is the content source indicator referring to the first name entry. Notice that the hierarchy of the content source indicators corresponds to the hierarchy of the access instructions and the data structure within the mass data storage application.

Figure 4 is a block diagram of the preferred insert statement parser rules 400 used by insert statement parser process 500 to parse an insert statement 205 into content source indicators 210. The left column contains keywords for the rules, and the right column contains the corresponding

indicators of these rules used in an insert statement 205. The following describes the rules one row at a time from top to bottom.

Insert statement identifiers 402 are "<" to indicate the possible start of an insert statement, and ">" to indicate the corresponding end of the insert statement. Next, order of parsing 405 refers to the order with which the insert statement parser process 500 parses an insert statement 205. In the case of a nested content source indicator 210, the order of parsing 405 applies to subsequent nested insert statements 205. In this preferred embodiment, parsing of nested insert statements takes place after the parsing of the insert statement containing it. The order of parsing 405 applies when there are more than one nested insert statement 205 within the content source indicator 210 of one insert statement 205. The blank character, or " ", is the preferred statement delimiter 415 which separates each state and condition statement. The hyphen character, or "-", is the preferred content source type indicator identifier 420 which identifies a content source type indicator 230 from content source indicators 210. In the examples, a content source type indicator 230 is the first content source indicator 210 of a state 225 or condition statement 245. The opening and closing parenthsis, or "(" and ")", is the preferred nested content source indicator identifier which identifies the beginning and end of a nested insert statement 205 respectively.

Figure 5 is an insert statement parser process flow chart 500 with start 505 and end 580 indicating the start and end of the process respectively. The process start is continued by a check for insert statement 510 within a source program file 900. Figure 2 further describes an insert statement 205. In this preferred embodiment, confirming the existence of an insert statement leads to an update of the insert statement context 515. Otherwise, a no to the check for insert

statement 510 leads to an exit from the insert statement parser process 500. Throughout this patent, a no to a conditional check is not explicitly marked in the diagrams and/or described in the writeup, then it leads to an exit from the insert statement parser process 500. Examples of this case include the conditional check on content source indicator 605, the conditional check on matched content source indicator 625, etc. The location of the updating of an insert statement context 515 can be anywhere between the start 500 and end 580.

An insert statement context 515 is stored in memory 145 and contains information which the insert statement parser process needs in order to continue its proper execution. For example, in the case of processing nested insert statements 205, the insert statement context 515 contains information on the recursive path such as a nested counter which keeps track of which layer of nested insert statement 205 is being processed, and the location of a nested insert statement 205 in the content source indicator 210 which contains it, etc.

Then the process continues by locating a logical condition statement 520, and parsing that statement 525. Notice that both a logical condition statement 245 and a state statement 225 could be parsed. Yielding a content source indicator leads to a check whether it is a nested content source indicator 530. The following describes first the subsequent steps when the content source indicator 210 is not a nested one, and then the steps when the content source indicator 210 is a nested one.

When a content source indicator 210 is not a nested one, it is inputted into the content source identification process 600, which identifies the content source object 305 to which the content

source indicator 210 is associated. Figure 6 further describes the content source identification process 600.

Once the content source object 305 is identified, the content source access process 700 follows the content source identification process 600 and accesses the content source object 305, after which there is check for end of that statement 545, whether it be a logical condition statement or a state statement. If it is not the end of that statement, then the insert statement parser process 500 continues with parsing the remaining statement 525. Otherwise, it continues with a check for an end of a logical condition statement 550. An end of a logical condition statement 245 leads to the evaluation of the logical condition statement 565. The preferred embodiment contains a binary evaluation and matching to the appropriate state statement 570, state one corresponding to the having access to the last content source indicator 210 of the logical condition statement 245, and state two corresponding to not having access to the last content source indicator 210. When multi-state evaluation is necessary, the preferred embodiment can be extended to contain an additional equality test on the content source object 305 through access the last content source indicator 210. Then the insert statement parser process 500 continues with locating 575 and parsing of the matched state statement 525.

On the other hand, not an end of a logical condition statement indicates the end of a state statement 225, and leads to the content insertion process 800. The content insertion process 800 retrieves the content from the content source object 305 accessed by the content access process 700, and inserts either the content or a reference to the content in place of where the evaluated

insert statement 205 is in the program source file 900. Figure 8 further describes the content insertion process 800.

Because the above description, as mentioned earlier, is on a content source indicator 210 which is not nested, the subsequent check on the content source indicator 210 being a nested one leads to the end 580. Notice that the end 580 could indicate either the end of the insert statement parser process 500, or end of parsing the nested insert statement 205, but not the end of the insert statement parser process 500. The following description on nested content source indicator 210 shall further justify the latter case.

Now consider the process when the check for nested content source indicator 530 indicates a nested one 210. The process performs a recursive call to the start 505 of the insert statement parser process, and proceeds again to the check for nested content source indicator 530. These steps cycle through until a content source indicator 210 checked is not a nested one, at which moment, the steps described for a non-nested content source indicator 210 above follow.

Notice that the subsequent updates on insert statement context 515, as in the case when there are nested content source indicators 210, would store in memory 145 information on the path from the preceding insert statement 205 to the steps involving the current insert statement 205 are recursively called from, as to be able to return the processing of the preceding one when the current one is processed. The logic behind the iteration and the corresponding access to CPU 165 and memory 145 are similar to that of a simple interactive program and the behavior of the stack and first-in-first-out queue.

To conclude, the interactive process of decomposing a nested content source indicator 210 to non-nested content source indicators 210 ends with the logical condition statement 245 and its corresponding state statement 225 of an insert statement 205 containing all non-nested content source indicators 210.

Figure 6 is a content source identification process flow chart 600. Once the insert statement parser process 500 determines a content source indicator 210 to be non-nested, the content source identification process 600 starts. First, it verifies whether it is indeed a content source indicator 605. Notice that this is a different check than that on whether a content source indicator 210 is nested because the latter focuses on the "nestedness". Then it further checks 610 whether a content source indicator is a content source type indicator 230.

In the case of a content source type indicator 210, the next step is to locate the content source access map 615. In the case of not a content source type indicator 210, the next step is to match the content source indicator 620 to the appropriate access instructions in the content source access map 615. If the content source indicator is matched 625, it's then onto the content source access process 700.

Figure 7 is a content source access process flow chart 700. The matched content source indicator 210 then leads to locating the access instruction to the content source object 705. Launching of the access instruction 710 then follows. Once gained access to the content source object 715, the flow of control goes back to the insert statement parser process 500.

Figure 8 is a content insertion process flow chart 800. Continuing from Figure 7, once the end of a state statement 245 is reached, the content insertion process 800 starts. It locates the content retrieval instruction to the content source object 805. Then it launches the retrieval instruction 810. Once content is retrieved 815, deciding to insert by value or reference 840 leads to either insert content by value 825 or by reference 830. Then the flow of control goes back to the insert statement parser process 500.

Deciding to insert content by value or reference 840 depends on many criteria. The type of a program source file is one preferred criterion. The type HTML of the example program source file 900 in Figure 9 is a criteria to deciding to insert by value or reference 840 on the insert statement, <INSERT -IMAGE.SKYEXBACKIMG -IMAGE.DEFAULTBACKIMG -TRANS.SKYEX> 905, which leads to inserting content by reference 830, specifically inserting a reference of the background image, back18.jpg 1005 as described in Figure 10.

Additional preferred criteria include the size of the content to be inserted, network bandwidth, occurrence rate of content within a program source file, update rate of the content within its content source object. The criteria could take the forms of threshold tests as follows:

Size: 1 Kilobits

The larger the size of the content, the more processing it takes to insert it by value relative to insert it by reference. The above criteria means that a content size below the threshold of 1 Kilobits goes into deciding for inserting by value, and for inserting by reference otherwise.

Network bandwidth: 1 Megabits per second

There are cases when content to be inserted is transmitted over a network 140. The above criteria means that a network bandwidth above the threshold of 1 Megabits per second goes into deciding for inserting by value, and for inserting by reference otherwise.

Occurrence rate: 10 times per program source file

10 insertions by reference could be associated with 1 copy of the content in the best scenario, while 10 insertions by value would lead to 10 copies of the content. The above criterion means that a content occurrence rate of 10 times or more per program source file goes into deciding for inserting by reference, and for inserting by value otherwise.

Update rate: 100 times per second

The higher the update rate, the more prone it is for corrupted or obsolete content which is inserted by value. The above criteria means that an update rate of 100 times or more per second to the content within its content source object goes into deciding for inserting by reference, and for inserting by value otherwise.

In the preferred embodiment, additional numerical weights are assigned to each criterion. Weights range from a preferred range of -10 to 10, with the negative range going into deciding for inserting by reference, and the positive range going into deciding for inserting by value. the greater the absolute value of a number, the greater the weight of that criterion in deciding to insert by value or reference 840. In addition, insertion by reference leads to different content depending on the type of a source program file. The example in Figure 9 illustrates this case.

Figure 9 is an example HTML source program file 900. In the example of an HTML source program file 900 included in this figure, three insert statements 205 highlighted in boldface and italic are embedded within regular HTML source. -TRANS.SKYEX is the logical condition statement 245 for the first insert statement, <INSERT -BK_IMAGE.SKYEXBACKIMG -BK_IMAGE.DEFAULTBACKIMG -TRANS.SKYEX> 905, and second insert statement, <INSERT -IMAGE.SKYEXLOGO -MAGE.DEFAULTBACKIMG -TRANS.SKYEX>910.

In a preferred processing of this example, this logical condition statement evaluates to true, therefore, an HTML reference to a background image for skyex 1005 is inserted as a result of evaluating the first insert statement 905, and an HTML reference to a skyex logo image 1010 is inserted as a result of evaluating the second insert statement 910. Notice that because the source program file is of type HTML, the HTML reference to the background image, ***background="image/back18.jpg"*** 1005 conforms to the HTML syntax by containing the word, ***background***. It also specifies the path to the file, ***"image/back18.jpg"*** properly for HTML rendering of it. Similarly, the HTML reference to the logo image, ***<img src="image/skyex.jpg">***

1010 conforms to the HTML syntax for rendering an image by containing the word, ***<img src=***. For source program files of other types, a reference of skyex.jpg could be sufficient.

-TRANS.USERCLAIMED is the logical condition statement for the third insert statement, <INSERT -TRANS.PERSON.COMPANY_NAME -HTML.DEFAULT_COMPANY_NAME -TRANS.USERIDCLAIMED> 915, and is evaluated true in a preferred processing of the example, therefore, -TRANS.PERSON.COMPANY_NAME is evaluated and a company name, U S Weather 1015, is inserted.

Figure 10 describes the HTML source after the three insert statements are resolved. Figure 11 describes the graphic rendering of the HTML source in Figure 10.

Figure 10 is an example rendering of an HTML source program file.
Following the above description for Figure 9, the first insert statement is resolved to back 18.jpg 1005. The second insert statement is resolved to skyex.jpg 1010. The third insert statement is resolved to U S Weather 1015.

Figure 11 is the resolved source of rendering of an example HTML program source file.
Following the above descriptions for Figure 9 and 10, the HTML rendering of <body background="image/back18.jpg"> 1005 results in the background color 1105 of the rendered image in this figure. The HTML rendering of <body background="image/back18.jpg"> 1010 results in the logo 1110 located to the left of the rendered image. The third insert statement is

resolved to U S Weather 1015 , the HTML rendering of which is U S Weather 1115, which is located to the right of the skyex logo.